Compiler Design HW1 Report

102062209 邱政凱

我的 Scanner 具備的功能基本上是依據 HW 的 SPEC 後再依據 C99 的 SPEC 在 Constant 的判斷上多增加了一些規則。基本上整體的運作邏輯是：吃到 token ->確認 token on/off, on 則輸出 token -> 塞到 buffer -> 遇到換行字符確認 source on/off, 若 on 則輸出 buffer 並清空。在 Error 的部分則是參考了 gcc 在-std=99 的設定下會把那些部分判斷成"lexical error"(而非 semantic error)，以確保不會處理到應該要是在 parser 階段才處理的部分，也不會漏處理應該要在 Lexical analysis 階段就判斷的 error。

Error 的輸出格式部分，看助教的範例好像助教的 code 是用 “/”運算子去看某些字元後面不能接哪些字元。而我認為輸出的 error 訊息要盡量完整，因此我是盡可能把 error 的 token 完整的吃進來用 yytext 輸出。(i.e. 01234 我會輸出 01234 而非 0，report 最後會有整理我的 error 輸出格式)

額外的部分我主要支援了一些 integer 和 floating point 的 suffix (123.123f、123L)和 string constant 的 prefix (L”123”)，在 double 的部分支援 123. 或 .123 這些支援在 C99 中的語法。並在 Pragma 宣告的部分完整參照 C99 的規則。

- Non-existent Character
  當輸入出現並沒有在 c99 中支援的字符時，我會直接輸出該字符作為 Error 訊息。
- Comment：吃到 “//” 後便進入 LINE Comment state，再吃到換行字元前的所有東西都直接放入 line buffer 而不輸出 token。吃到” /*” 則進入 COMMENT state，在遇到另一個” */” 前把所有吃到的東西都放到 line buffer(不過遇到\n 要換行)。
- Pragma : 基本上我用了比較全面的規則來處理 pragma 的部分

```
^{BLANK}*#{BLANK}*pragma{BLANK}+source{BLANK}+on{BLANK}*("//".*)?   {srcOn = 1;PutToBuffer(yytext);}
^{BLANK}*#{BLANK}*pragma{BLANK}+source{BLANK}+on{BLANK}*("/*".*)?   {srcOn = 1;PutToBuffer(yytext);BEGIN COMMENT;}
^{BLANK}*#{BLANK}*pragma{BLANK}+source{BLANK}+off{BLANK}*("//".*)?      {srcOn = 0;PutToBuffer(yytext);}
^{BLANK}*#{BLANK}*pragma{BLANK}+source{BLANK}+off{BLANK}*("/*".*)?      {srcOn = 0;PutToBuffer(yytext);BEGIN COMMENT;}
^{BLANK}*#{BLANK}*pragma{BLANK}+token{BLANK}+on{BLANK}*("//".*)?            {tokenOn = 1;PutToBuffer(yytext);}
^{BLANK}*#{BLANK}*pragma{BLANK}+token{BLANK}+on{BLANK}*("/*".*)?            {tokenOn = 1;PutToBuffer(yytext);BEGIN COMMENT;}
^{BLANK}*#{BLANK}*pragma{BLANK}+token{BLANK}+off{BLANK}*("//".*)?           {tokenOn = 0;PutToBuffer(yytext);}
^{BLANK}*#{BLANK}*pragma{BLANK}+token{BLANK}+off{BLANK}*("/*".*)?           {tokenOn = 0;PutToBuffer(yytext);BEGIN COMMENT;}
#.*         {printf("No Matched pragma rule\n");Error();}
```

Pragma 依照 C99 的 STANDARD 必須要:
在同一行的前面不能有空白(含 comment)以外的字元、每個單字之間至少都要有空白(含 comment)間隔、Pragma 完整敘述後不能再加除了 comment 以外的其他敘述。因此我的 scanner 在 pragma 敘述不符合以上三種狀況，或是打了不存在的 pragma 時會輸出 error。

例如：

123 #pragma source on 、 #pragma source once 、 # pragma source on x = 123 我都會輸出成 error。

而我的判斷正確的 pragma 方式就是# 、 pragma 、 source/token 、on/off 全部都必須同時出現，且它們之間必須都間隔一個以上的空白或/**/comment。

- Identifier:

  C99 中定義的 identifier 的合法字元包刮 0-9 a-z A-Z 和_ (底線)。

  而一個合法的 identifier 只要第一個字元不是 0-9，其他各種各樣的組合都是合法的。 E.g. __123 、 a12__。所以我處理 identifier 的方式基本上就是[a-zA-Z_][0-9a-zA-Z_]*

- KEY : 基本上 C99 定義的關鍵字我就是直接把助教的投影片上列舉的 KEY 打上去並直接輸出成 KEY 的 Token。Stdio.h 的部分則是另外去找 stdio.h 把裡面的 library function 照著字典序排列輸入。

- Integer constant

```
0{ISUFIX}?{digit}+              {Error();}
[1-9]{digit}*{ISUFIX}?      {PutToBuffer(yytext);PrintToken(INTEGER,yytext);}
0{ISUFIX}?                  {PutToBuffer(yytext);PrintToken(INTEGER,yytext);}
```

  Integer constant 的部分，我支援了 C99 的 integer suffix(123U,123LL)來描述 unsigned 或 long 的狀況。在 error 的部分我會在第一個數字是 0 而有其他數字接在後面的狀況發生。

  合法:

  (1) 0

  (2) {Non-zero digit} {digit}

- Double constant:

```
{digit}+\.{digit}*{FSUFIX}?     {PutToBuffer(yytext);PrintToken(DOUBLE,yytext);}
{digit}+\.{digit}*{FSUFIX}?(\.|_|{ad})+     {printf("double Error\n");Error();}
\.{digit}+{FSUFIX}?         {PutToBuffer(yytext);PrintToken(DOUBLE,yytext);}
\.{digit}+{FSUFIX}?(\.|_|{ad})+     {printf("double Error\n");Error();}
```

  Double constant 的部分我除了支援基本的 123.123 這種形式外還支援了 .123 和 123. 這些省略 0 的表示方式。並支援 floating point constant 的 suffix (f、F、l 、L) 。另一點比較值得說明的就是，在 C99 standard 中，double 的整數部分是可以有 0 當作 prefix 的，例如: 00123.456 在 C99 的 standard 中是合法的，因此如果判斷是 double，我就不會針對開頭是 0 這件事輸出 error。

  合法:

  (1) {digit sequence}(opt) . {digit sequence}{floating point suffix}(opt)

  (2) {digit sequence} . (nothing) {floating point suffix}(opt)

  而 Error 則會在 double 的敘述後還有接 “\.” 、[:alpha:]、 “_” 等等字元的

狀況下發生。一個 token 連接這些字符通常 lexical analyzer 會判斷為同一個 token 的延伸，而非不同的 token。這部分我會這樣判是因為我觀察 gcc 的 compiler，像是 123.123.123、 123.123abc、 123.123_x_ 等等的狀況下，gcc 會輸出 “too many floating points” 、 “inappropriate suffix”等等的敘述，因此可以推斷應該是 lexical error，而非文法階段的 error，所以我才把這些規則納入。

- Scientific Notation

```
{digit}+\.{digit}*[eE][+-]?{digit}+{FSUFIX}?    {PutToBuffer(yytext);PrintToken(SCI,yytext);}
{digit}+\.{digit}*[eE][+-]?{digit}+{FSUFIX}?(\._|{ad})+    {printf("SCI Error\n");Error();}
\.{digit}+[eE][+-]?{digit}+{FSUFIX}?        {PutToBuffer(yytext);PrintToken(SCI,yytext);}
\.{digit}+[eE][+-]?{digit}+{FSUFIX}?(\._|{ad})+        {printf("SCI Error\n");Error();}
{digit}+[eE][+-]?{digit}+{FSUFIX}?          {PutToBuffer(yytext);PrintToken(SCI,yytext);}
{digit}+[eE][+-]?{digit}+{FSUFIX}?(\._|{ad})+      {printf("SCI Error\n");Error();}
```

科學記號的部分我主要也是參考 C99 的 standard。基本上 c99 定義 scientific notation 就是 {合法的 FP constant}[eE][+-]?{digit}+ 這樣的形式，只是 suffix 必須放到最後面。

合法:

(1): {digit sequence}(opt) . {digit sequence} [eE] [+-] {floating point suffix}(opt)

(2): {digit sequence} . [eE] [+-] {digit sequence}{floating point suffix}(opt)

(3) {digit sequence}[Ee] [+-] {digit sequence}{floating point suffix}(opt)

Error 的部分規則類似 double，就是在合法的 scientific notation 後馬上接 “\.” 、[:alpha:]、 “_” 等等字元的狀況下發生。至於 123e (後面沒接數字)、123e*123(非+-的記號)等等的狀況，因為 “123e” 這樣的規則本身就是個非法的 integer constant(或非法的 identifier)，因此在我的 scanner 中這兩種狀況在他被認定成是 scientific notation 的 error 之前它就會被別的地方判掉了

- Character constant

字元常數的部分依照 C99 的定義，不能是以下三種形式 ‘\’ 、 ‘\n’ 、 ‘\’’ 也就是不能有單引號、換行字符、或是單斜線的出現。

- String Consant

String constant 比較特別的地方是在 token 的輸出需要把雙引號拔掉，在 print source 的部分則不需要，因此用以下這段 code，先把 yytext 直接塞進 buffer，再把雙引號拔掉後 print token 出來。

```
L?\"{STRCH}*\"      {
                        PutToBuffer(yytext);
                        char* temp;
                        if(yytext[0]=='L'){
                            temp = (char*)malloc(sizeof(char)*(strlen(yytext)-3));
                            int i=0;
                            for(i=0;i<strlen(yytext)-3;i++)
                                temp[i] = yytext[i+2];
                        }
                        else{
                            temp = (char*)malloc(sizeof(char)*(strlen(yytext)-2));
                            int i=0;
                            for(i=0;i<strlen(yytext)-2;i++)
                                temp[i] = yytext[i+1];
                        }
                        PrintToken(STRING,temp);
                    }
```

例外根據 c99，跟 character constant 有點像，只是單引號是被允許的，而雙引號是不被允許的，因此 match 的 rule 吃入兩個雙引號中夾著的所有排除 '\' 、 '\n' 、 '\"' 的字元。例如 "123\q123" (不存在的跳脫字元)、 "123 <EOF> (未匹配的雙引號) 等等我都會輸出成 ERROR。

Error Handling:

以下列舉我額外處理的 error 以及範例的 input 和對應輸出的 error：

	Input	output
0{digit}+{ISUFIX}?	01234	01234
{digit}+({alpha}|_)({alpha}|{digit}|_)*	123abc	123abc
	123e+	123e
	123e*321	123e
{digit}+\.{digit}*{FSUFIX}?(\.|_|{ad})+	123.123abc	123.123abc
	345.abc	345.abc
	123.123.123	123.123.123
	123..123	123..123
\.{digit}+{FSUFIX}?(\.|_|{ad})+	.123abc	.123abc
{digit}+[eE][+-]?{digit}+{FSUFIX}?(\.|_|{ad})+	1E+3abc	1E+3abc
\.{digit}+[eE][+-]?{digit}+{FSUFIX}?(\.|_|{ad})+	.1E+3abc	.1E+3abc
{digit}+\.{digit}*[eE][+-]?{digit}+{FSUFIX}?(\.|_|{ad})+	2.2E+3abc	2.2E+3abc
^{BLNAK}*#(_|{alpha}|{digit})*	#pragma source once	#pragma source once
	#include <stdio.h>	#include